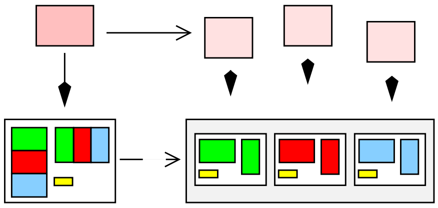
ADAPTOR
Users Guide
Version 6.1 (Dec 1998)

T. Brandes
R. Höver-Klier



Report Adaptor 2

December 21, 1998

GMD

SCAI

GMD – German National Research Center for Information Technology
Institute for Algorithms and Scientific Computing, SCAI
Schloss Birlinghoven
D-53754 Sankt Augustin
Germany

Tel.: +49 (0)2241 / 14-2492
E-mail: brandes@gmd.de, hoever-klier@gmd.de

This page is left empty

ADAPTOR
Users Guide
Version 6.1 (Dec 1998)

T. Brandes
R. Höver-Klier

GMD – German National Research Center for Information Technology
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany

Contents

Abstract

ADAPTOR (Automatic DAta Parallelism TranslatOR) is a public domain High Performance Fortran (HPF) compilation system developed at the SCAI institute (GMD) during the last years. The tool transforms data parallel programs written in Fortran with array extensions, parallel loops, and layout directives into programs with explicit message passing. The latest version incorporates the feedback of many users and contains already a lot of optimization techniques. Beside some restrictions, it supports the new standard HPF 2.0 as well as many of the approved extensions within this new standard.

This users guide describes how the ADAPTOR tool can be used and which effects the different compiler flags will have. It also explains how to run the data parallel HPF programs as serial programs and as parallel message passing programs.

1 Overview

ADAPTOR consists of three components:

- the source to source transformation tool fadapt,
- the distributed array library DALIB is the HPF runtime system that handles descriptors for arrays, sections and distributions and implements communication routines,
- the compiler driver gmdhpf uses cpp and `fadapt` to compile HPF programs into FORTRAN 77 or Fortran 90 code with explicit message passing, then invokes a native Fortran compiler upon the users choice to complete the compilation process and finally invokes a suitable link step.

Figure 1 shows how `fadapt` and DALIB work together. The tool `fadapt` generates message passing programs that will be compiled with a native Fortran compiler and linked together with the DALIB. The generated executable can easily be started on the parallel system.

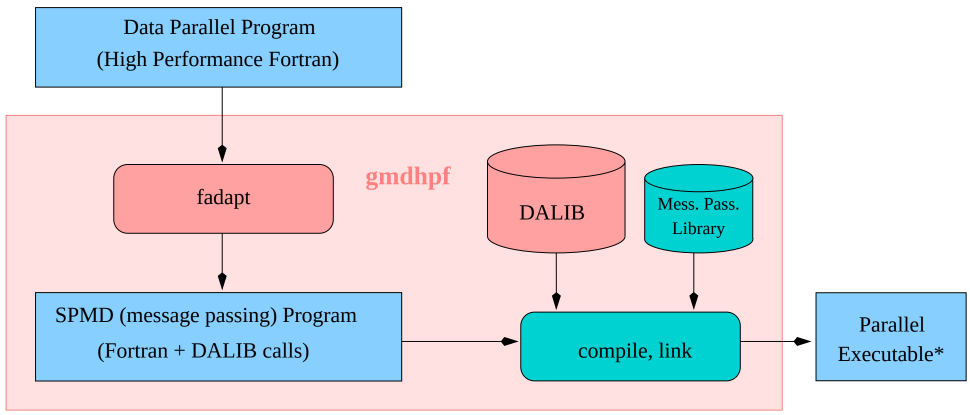


Figure 1: Overview of the ADAPTOR tool

Before continuing, make sure that ADAPTOR has already been installed on your machine (see also ADAPTOR installation guide [BHK98]) and that you know in which directory it is installed. Although you can make your own installation, it is recommended to make a system-wide installation for all users.

A detailed description of the input language for ADAPTOR can be found in the ADAPTOR programmers guide [Bra98]. There are also many example programs available that can be used to study the functionality of the compilation system.

2 Setting the Environment

Let be `<install-dir>` the directory on the machine where ADAPTOR is installed. Every user should set the environment variable `PHOME` to this directory in the following way:

```
setenv PHOME <install-dir>   or    export PHOME=<install-dir>
```

Furthermore, the `bin`-directory of ADAPTOR should be included in the path variable, the `man`-directory in the manpath variable.

```
setenv PATH $PHOME/bin:$PATH
setenv MANPATH $PHOME/man:$MANPATH
```

Now there should be no problem to call the commands of ADAPTOR, e.g. the following commands should work correctly:

```
fadapt -help
gmdhpf -help
man fadapt
man gmdhpf
```

Furthermore, make sure that the DALIB runtime system is available. The following files should exist:

```
$PHOME/dalib/<MP>/dalib.a
$PHOME/dalib/<MP>/unilib.a
```

where `<MP>` stands for the appropriate message passing system on your machine (e.g. `PVM3`, `MPI`). Beside the DALIB, the UNILIB is used for linking HPF programs compiled for a single node.

3 About the new Version

The ADAPTOR tool is already available for a longer time and there have already been released some earlier versions. Before starting make sure that you have the right version of the source-to-source translation `fadapt` and the DALIB runtime system.

```
fadapt -version
```

It should be version 6.1 otherwise this user manual might be incorrect for some features.

Attention: It is absolutely necessary that the DALIB runtime system has the same version as the source-to-source translation. Otherwise there might be inconsistencies between the generated DALIB calls and the latest interfaces of these routines.

```
what $PHOME/dalib/<MP>/dalib.a
```

The latest version of ADAPTOR supports more HPF features than the previous ones and has less restrictions. The following functionalities have been added during the last time:

- Certain flags of fadapt can be set directly in the source file (see section 5.23).

- Different data parallel HPF tasks can run in parallel and exchange data with the new `HPF_TASK_LIBRARY`. The runtime system supports the multile execution of different HPF tasks (see section 9).
- The source-to-source translation is able to identify serial loops that are `INDEPENDENT` (see section 5.14).
- The source-to-source translation can generate code that is better suited for automatic vectorization (see section 5.15). This results in better performance for vector architectures (e.g. NEC SX/4).

4 Testing the Compilation

4.1 The Example Program

The following data parallel HPF program `prime.hpf` (assumed to be in the directory `$PHOME/hpf_examples/simple/`) computes the number of primes in the range from 2 to n. The program uses dynamic arrays and array syntax. Timing functions are used to measure the run time of the program.

There is exactly one array in the program. This array will be block distributed among the nodes.

```
      program prime
      integer n, s, k
      logical, allocatable :: a(:)
!hpf$ distribute a(block)
      integer tickstart, tickstop, tickrate
      print *, 'Input n for counting primes in range 2 to n : '
      read *, n
      call system_clock (tickstart, tickrate)
      allocate (a(1:n))
      a = .true.
      a(1) = .false.
      k = 2
      do while (k*k .le. n)
         a(k*k:n:k) = .false.  !  sieve all multiples of k
         k = k + 1
         do while (.not. a(k))   ! find next prime k
           k = k + 1
         end do
      end do
      s = count (a)
      call system_clock (tickstop)
      tickstop = tickstop - tickstart
      print *, 'There are ',s,' primes until ', n
      print *, 'Time needed : ', float(tickstop)/float(tickrate)
      deallocate (a)
      end
```

4.2 Basic Compilation

The compile driver `gmdhpf` drives the whole compilation and will generate a corresponding executable. The compile driver is described in detail in section 6.

```
gmdhpf -o <executable> <file>.hpf
```

This call will generate the executable in three steps:

- In the first step, the data parallel program will be translated to a Fortran program by the source-to-source translation `fadapt` (see Section 5).

- The generated Fortran program will be compiled by a native Fortran Compiler. Since options for this compilation are compiler dependent, we refer to the corresponding Users Guide.
- The compiled program will be linked with the HPF runtime system DALIB and the corresponding message passing library.

The following translation will generate a parallel SPMD program.

```
gmdhpf -o prime prime.hpf
```

4.3 Compiling for a Single Node

The following command translates the data parallel program to a serial Fortran program, and will then compile and link it.

```
gmdhpf -1 -o prime prime.hpf
```

The program runs only on a single node, it contains no message passing.

4.4 Compiling for a Single Multiprocessor Node

The following command translates the data parallel HPF program to a Fortran program containing multiprocessor parallelization directives (e.g. OpenMP directives [SGI97]), and will then compile and link it.

```
gmdhpf -1 -mp -o prime prime.hpf
```

Attention: This feature is still in development and only available for certain machines.

5 The Source-to-Source Translation fadapt

This section describes the use of the source-to-source translation and all the flags that can be used to drive the translation. An overview of all options is given in the appendix in table 2.

5.1 Call of fadapt

With the following command the sources of a data parallel program are directly translated to a message passing program.

```
fadapt <source_file_names>
```

More than one source file may be specified. All source files together stand for one data parallel program. There must be exactly one main program in the specified source files. All subroutines and functions that are not defined are assumed to be external HPF routines.

`fadapt` will accept files with the following suffixes as valid input source files:

```
.f    .f9      .f90     .hpf
```

Attention: Files ending with the suffix `.f` will not be passed to `fadapt` by the compile driver `gmdhpf` (see section 6.4). But they can be translated directly by `fadapt`.

If the translation is executed correctly, the following files will have been generated by default:

```
cube.f          (SPMD node program)
node1.f         (serial program)
```

With the option `-f` it is possible to specify an own name for the generated output file.

```
fadapt <source_files> -f <output_file>
```

5.2 Protocol Files of Translation

`fadapt` generates protocol files according to the translation steps. By default, `fadapt` removes silently all these files except a warning or an error message have occurred in the translation step.

These files are very useful in situations where errors have been encountered. If an error occures within one phase, the translation will stop immediately after this phase.

The option `-list` will keep all intermediate protocol files.

```
fadapt -list
```

The following protocol files will be generated:

```
adaptor.def      (protocol file of semantic phase 1: make defs)
adaptor.sem      (protocol file of semantic phase 2: sem checks)
adaptor.cf       (protocol file of semantic phase 3: control flow)
adaptor.dist     (protocol file of parallel phase 1: distribution)
adaptor.normal   (protocol file of parallel phase 2: normalization)
adaptor.auto     (protocol file of parallel phase 3: loop parallelization)
adaptor.ovl      (protocol file of parallel phase 4: shadows)
adaptor.ana      (protocol file of parallel phase 5: analysis)
adaptor.arg      (protocol file of adaptor phase 1: arguments)
adaptor.loop     (protocol file of adaptor phase 2: make loops)
adaptor.home     (protocol file of adaptor phase 3: home determination)
adaptor.local    (protocol file of adaptor phase 4: communication generation)
adaptor.class    (protocol file of adaptor phase 5: classification)
adaptor.opt      (protocol file of optimization phase 1: optimization)
adaptor.ipa      (protocol file of optimization phase 2: interprocedural)
adaptor.code     (protocol file of codegen phase 1: code generation)
adaptor.addr     (protocol file of codegen phase 2: addressing)
adaptor.final    (protocol file of codegen phase 3: final code)
```

These files can be deleted with the command $\boxed{\text{adapt.clean}}$.

5.3 Intermediate Files of Translation

It is possible to generate intermediate files of the source-to-source translation. This might be very convenient to understand in detail how the compiler has translated the input file and has taken advantage of the available data parallelism.

```
fadapt -debug
```

The following files will be generated after the corresponding translation steps:

```
cf_unparse.f     (source program after semantical analysis)
normal_unparse.f (source program after normalization)
ana_unparse.f    (source program after arguments)
arg_unparse.f    (source program after arguments)
loop_unparse.f   (source program after loop generation)
home_unparse.f   (source program after home determination)
local_unparse.f  (source program after localization)
opt_unparse.f    (optimized source program before code generation)
code_unparse.f   (code generation for DALIB calls)
addr_unparse.f   (localizing addresses and pseudo dynamic arrays)
```

These files can also be deleted with the command adapt.clean.

5.4 Verbose and Help Information

The two flags `-help` and `-settings` are very useful to get information about the available options and about the default options of `fadapt`. The flag `-help` displays the list of all available options while the flag `-settings` prints the list of the default options. The default options depend on the installation and might vary for the different machines.

```
[-help]            list all available options
[-settings]        prints the list of the set options
```

The verbose flag `-v` should be used to get information about the progress of `fadapt`. The verbose mode is also switched on with the flag `-G` that is an abbreviation for the three flags `-v`, `-list` (see section 5.2) and `-debug` (see section 5.3).

```
[-v]               verbose
[-debug]           do not delete intermediate files
[-list]            do not delete protocol files
[-G]               combines -v -debug -list
```

5.5 Warnings and Info Messages

By default, fadapt prints warning and information messages. They can be switched off by the following options:

```
[-w]               does not print any warnings
[-noinfo]          does not print any info messages
```

5.6 Choosing the Execution Model

The user can select between two programming models:

- `SPMD` (single process, multiple data) will generate a program that runs on all available nodes. There is no host program. The first node takes care of all I/O operations.

- SERIAL will generate a program that runs on a single node. It has no communication and therefore it ought to be faster than the SPMD version running on a single node.

For the call of fadapt the execution model can be specified by the following options:

```
-N         model = HPF_GLOBAL    (single program, multiple data)
-1         model = HPF_SERIAL    (single program, single data)
```

Note: With the flag -1 fadapt handles all routines of the source file like routines with the prefix EXTRINSIC(HPF_SERIAL) (see programmers guide [Bra98]).

5.7 Choosing the Source Language

ADAPTOR supports nearly full High Performance Fortran 2.0 [Hig97]. As other HPF compilers might only support the base language or even the HPF subset only, the user can verify by a corresponding flag that only the corresponding featurs are used.

```
-HPF_SUBSET     source language = HPF (subset)
  -HPF_BASE     source language = HPF 2.0 (base language)
       -HPF     source language = HPF 2.0 (full language)
   -HPF_ADP     source language = full HPF 2.0 + ADAPTOR extensions
```

ADAPTOR provides some specific features (e.g. the TRACE directive, the SELECT directive, ARBITRARY distributions) that are not HPF standard. They are only permitted when the full language with ADAPTOR extensions has been selected. When using the flag -HPF_SUBSET instead of -HPF, fadapt will verify that only the HPF subset is used.

When the flag -nostrict (see section 5.11) is enabled, only warnings are given. Otherwise any violation will produce an error and stop the translation.

Note: Previous versions of ADAPTOR supported also Connection Machine Fortran. Since version 5.1, this language is no longer supported.

Note: ADAPTOR still has some restrictions for compiling HPF programs. HPF programs that violate these restrictions will not be accepted at all. In this case, the user has to modify his code to fullfil the restrictions (e.g. own redistribution of data and extracting communication out of independent loops on his own).

5.8 Choosing the Target Language

For portability reasons, the general strategy of fadapt is to generate FORTRAN 77 programs. Therefore, the array operations are translated to serial loops and the dynamic arrays will be simulated by pseudo-dynamic arrays in the generated programs.

Until now, some Fortran 90 features cannot be translated to FORTRAN 77 by fadapt. This includes especially modules, contained subroutines, generic procedures, overloading of operators, derived types, and pointers. If one of these features is used, the target compiler must be a Fortran 90 compiler.

```
-F77        FORTRAN 77
-F90        Fortran 90
```

Attention: The functionality of the input language is restricted if the target language is not Fortran 90. If the flag -F77 is set, fadapt will check that the generated programs can still be compiled by a FORTRAN 77 compiler.

5.9 Default Distributions

If the user does not give any layout or distribution directive for an array in his data parallel program, a default distribution will be chosen. The following options are intended for selecting a strategy for the default distribution (a more detailed description can be found in the Programmers Guide [Bra98]).

Scalar variables will always be replicated.

```
  -ddr      default distribution of arrays is replicated
  -ddb      default distribution of arrays is block distribution
            along the last dimension
```

5.10 Maximal Number of Distributed Dimensions

Although a data parallel program has been written that can take advantage of many distributed dimensions, it might be useful for smaller machines to distribute only one or two dimensions. The generated code will usually be faster.

```
  -d <n>    specifies the maximal number of distributed dimensions
```

The effect of this option is that every distributed array with more than n distributed dimensions will only be distributed along the last n of these dimensions.

```
      real A (n,n,n)
!hpf$ distribute A(block,block,block)
      ....
```

For `-d 1` this directive will be treated as follows:

```
      real A (n,n,n)
!hpf$ distribute A(*,*,block)
      ....
```

5.11 Semantical Checks

During the semantic analyis, `fadapt` checks whether the number of actual arguments is equal to the number of dummy arguments. Also the correct rank of arrays and the correct types are checked.

```
      real A(6,6), B(5)
      ...
      call SUB (A, B)
      ...
      subroutine SUB (X, Y, Z)
      real X(36), Z
      integer Y(5)
      ...
      end
```

This example might give at least 3 semantical errors:

- rank of actual `a` mismatches rank of dummy `x`

- type of actual `b` mismatches type of dummy `y`
- the actual call has only two arguments instead of the three expected ones.

If the flag `nostrict` is switched on, only warnings will be given and the translation is continued.

```
[-nostrict]
```

`fadapt` is more restrictive with its semantical checks than most FORTRAN 77 compilers usually are. The flag disables serious error messages for this kind of errors. But the user should be careful in any case as there might arise other problems, especially for distributed arrays.

5.12 Runtime Checks

At runtime, the distribution of the actual arguments will be matched against the distribution of the dummy arguments. By default, a redistribution takes place if there is a mismatch.

As in many user programs no redistributions take place at all, some additional overhead for matching distributions can be avoided. `fadapt` provides the following switches:

```
[-safety 2]     ! runtime checks, does redistributions if necessary
[-safety 1]     ! runtime checks, but no redistribution
[-safety 0]     ! no runtime checks
```

The default value is `safety 2`.

Attention: If the safety value is 0 or 1, there will be no redistributions at subroutine boundaries. In case of `safety 1` there are runtime checks that verify correct distributions of the dummy arguments. In case of `safety 0`, there are no checks at all, the generated programs will contain less calls to the runtime system.

```
      subroutine SUB (A,N)
      integer N
      real A(N,N)
!hpf$ distribute A(block,block)
      ...
      end subroutine SUB
      ...
      real A(N,N), B(N,N)
!hpf$ distribute A(block,block)
!hpf$ distribute B(cyclic,cyclic)
      ...
      call SUB (A,N)   ! always okay as no redistribution necessary
      call SUB (B,N)   ! redistribution with -safety 2
                       ! runtime error with -safety 1
                       ! unpredictable behavior with -safety 0
```

Note: The runtime checks will always be done in the called routine. There a no checks within the calling routine.

5.13 Compiler Optimization

A lot of compiler optimizations have already been implemented (e.g. automatic detection of overlap areas, inlining of cshift, loop fusion).

```
[-O]            ! optimization is switched on
[-noopt]        ! no optimization
```

By default, all optimizations are switched on.

5.14 Automatic Loop Parallelization

`fadapt` can identify independent `DO` loops. Independent loops that have not the `INDEPENDENT` directive will be considered like independent loops. Furthermore, scalar reduction variables are identified properly.

```
[-auto]          ! tries to identify INDEPENDENT do loops
[-noauto]        ! disables automatic loop parallelization
```

If the automatic loop parallelization is switched on, ADAPTOR will also inform about loops that could not be proven to be `INDEPENDENT`.

5.15 Vectorization

`fadapt` can generate code that is better suited for automatic vectorization of the native Fortran compiler. This feature is only supported for the NEC SX-4 and not available in the public domain version.

```
[-vector]          ! generates better vectorizable code
[-novector]        ! generates usual code
```

If the vectorization is enabled, `fadapt` acts on the new `SELECT` directive and moves vectorizable loops to inner loops.

5.16 Shared Arrays

ADAPTOR supports shared arrays (see [Bra97]). Distributed arrays can be made shared by default with the following flag:

```
[-shared]    ! distributed arrays without any replication are shared \\
```

5.17 Static Arrays

Common arrays and local arrays of a fixed size are often considered to be static. As they become dynamic as soon as they are distributed, a special flag is needed to let these arrays remain static.

```
[-static]   &  static distributed arrays will not become dynamic \\
```

5.18 Multiprocessor Parallelization

fadapt can generate code that contains explicit directives for multiprocessor parallelization.

```
[-mp=no]        ! generates usual code
[-mp]           ! generates Open MP parallelization directives
[-mp=sun]       ! generates SUN MP parallelization directives
[-mp=sgi]       ! generates SGI MP parallelization directives
[-mp=cray]      ! generates CRAY MP parallelization directives
```

This feature is still in development and might not work well currently.

5.19 Relevant Columns in Input Files

Since fadapt is very flexible in handling Fortran sources for input and output, the user may choose between the fixed and free source format. For the fixed source format the user may specify also the number of relevant columns of the input file.

```
[-fix <n>]         number of relevant columns in input file
[-free]            free source format for input file
```

5.20 Relevant Columns in Output File

The generated output files are always in fixed format. The number of relevant columns for the output file is very important as it must be set according to the facilities of the used native Fortran compiler. Otherwise a typical error message like the following one will appear:

```
Line 162  Error message # 1139
  Perhaps missing RPARENT before ENDOFST
   --  [unexpected end of statement]
```

If this happens, a smaller number of column characters must be specified by using the strip option.

```
[-strip <n>]       number of relevant columns for output file
```

If the target compiler supports extended source lines, the value 132 for n should be used in favor of getting better readable output files. If not, the value 72 should be tried as this is the default number of relevant columns for Fortran sources in fixed format.

```
! -strip 72
 call BOX (NELX,NELZ,X(X_ZERO+DALIB_local_lbound(X_DSP,2)*X_DIM1+1)
&,IX(IX_ZERO+DALIB_local_lbound(IX_DSP,2)*IX_DIM1+1),NUMNP,
$  NUMEL2,DALIB_0,DALIB_0,X_DSP,IX_DSP,DALIB_0,DALIB_0)
```

5.21 Word Lengths

For the correct generation of message passing programs it is very important that fadapt knows about the default size of integer values, real values and addresses.

```
INTEGER I, J
REAL C, D
```

Usually integer values will require 4 bytes and real values will also require 4 bytes. On some machines only a word length of 8 bytes is supported (e.g. CRAY, KSR1).

No problems should occur if integer, real and address length for a FORTRAN program are identical to the values of `sizeof(int)`, `sizeof(float)`, and `sizeof(*int)` called from a C program. In this case, the DALIB will work correctly with the generated programs.

But there might arise some problems if the ADAPTOR compilation system is used as a cross compiler and not directly on the target machine. Then the following flags must be used to set the correct values.

```
-i4          ! default integer length is 4 bytes
-i8          ! default integer length is 8 bytes
-r4          ! default real length is 4 bytes
-r8          ! default real length is 8 bytes
-a4          ! default size of one address is 4 bytes
-a8          ! default size of one address is 8 bytes
```

On the Cray T3D the cf77 compiler handles REAL values like double values in C. In this case, the following flag is absolutely necessary:

```
-dp          ! REAL has to be handled like double precision
```

Note: There will be in any case a verification of correct sizes at runtime.

5.22 Include Directories

With the option `-I<directory_name>` additional directories can be specified where include files and module definition files are searched for. Without any specification, only the current working directory is searched.

```
program RUN
use M              ! module M defined in /home/user/x2/modules/M.hpf
'include data.h'   ! is file /home/user/x2/include/data.h
...
end program
```

This program should be compiled with the following options:

```
fadapt -I/home/user/x2/modules -I/home/user/x2/include ...
```

5.23 Options in Source File

Many of the flags for the source-to-source translation can be overwritten directly in the source file that is compiled. This approach makes sure that one certain file or one certain routine is always compiled in the same way.

```
!ADP$ FLAGS -auto -no_vector -mp -free
```

This approach is also very useful when different source files of one application should be compiled in different ways.

The following options can be changed:

- The number of relevant columns for fixed source format or the free source format can be specified (see section 5.19).
- Automatic loop parallelization can be switched off or on.
- The optimization level can be set.
- Shared memory parallelization
- vectorization
- safety
- semantical checks
- source language.

All other options cannot be changed in this way.

5.24 Changing Default Options

With the following command one can get a summary of the default options.

```
fadapt -settings
```

An alias can be defined to set its own default options.

```
alias fadapt1 'fadapt -fix 72 -strip 72 -N \!*'
```

Afterwards the command `fadapt1` can be used like `fadapt` but with different default options.

```
fadapt1 -settings

Default Distribution : replicated  [-ddr]
Source Language      : HPF (subset) [-HPF_SUBSET]
Target Language      : Fortran 77 [-F77]
Target Model         : SPMD prog [-N]

default integer size : 4
default real    size : 4
default address size : 4

Maximal number of distributed dims : 3

Optimization is switched on

Safety level is                    : 2

Relevant columns for input sources : 72
Strip length for generated sources : 72

Verbose = 0,   Debug = 0
```

5.25 INTERFACE blocks

Beside the full compilation, `fadapt` can generate a call graph (see section 5.26) and an interface file.

For separate compilation it might be necessary that a subroutine or a function contains an `INTERFACE` block that specifies the parameter lists of all the subroutines or functions that are called.

For existing codes it might be rather tedious to insert corresponding `INTERFACE` specifications. A good solution is to use an interface file `interface.h` that will be included in all subroutines.

ADAPTOR can generate such an interface file automatically from all the given source files.

```
fadapt -interface *.f
```

This command generates a file `interface.h`. It contains an `INTERFACE` block with all subroutines of all source files.

```
INTERFACE
SUBROUTINE DIFMX (U, TX, TY, NP, T, MXRESUL)
INTEGER*4 NP
REAL*8 U (0:NP,0:NP)
REAL*8 TX (0:NP,0:NP)
REAL*8 TY (0:NP,0:NP)
REAL*8 MXRESUL
INTEGER*4 T
END

SUBROUTINE DIFL2 (U, TX, TY, NP, T, L2RESUL)
INTEGER*4 NP
REAL*8 U (0:NP,0:NP)
REAL*8 TX (0:NP,0:NP)
REAL*8 TY (0:NP,0:NP)
REAL*8 L2RESUL
INTEGER*4 T
END

SUBROUTINE SOL (U, TX, TY, NP, T)
INTEGER*4 NP
REAL*8 U (0:NP,0:NP)
REAL*8 TX (0:NP,0:NP)
REAL*8 TY (0:NP,0:NP)
REAL*8 T
END

SUBROUTINE F1 (U, TX, TY, NP)
INTEGER*4 NP
REAL*8 U (0:NP,0:NP)
REAL*8 TX (0:NP,0:NP)
REAL*8 TY (0:NP,0:NP)
END

SUBROUTINE F2 (U, TX, TY, NP)
INTEGER*4 NP
REAL*8 U (0:NP,0:NP)
REAL*8 TX (0:NP,0:NP)
REAL*8 TY (0:NP,0:NP)
END

SUBROUTINE G (U, TX, TY, NP, T)
INTEGER*4 NP
REAL*8 U (0:NP,0:NP)
REAL*8 TX (0:NP,0:NP)
REAL*8 TY (0:NP,0:NP)
REAL*8 T
END

END INTERFACE
```

5.26 Call Graph

fadapt can also be used to generate a call graph for a given Fortran program. The call graph of a program gives information about which subroutine calls which other subroutine. It can be generated in the following way:

```
fadapt -call *.f
```

The file test.call will contain the following information:

```
CALL Hierarchy (User Nodes)
===========================
EXPL2
 . F1
 . SOL
 . DIFMX
 .  . SOL (+)
 . DIFL2
 .  . SOL (+)
 . G
```

This information should also be used to place the INTERFACE blocks correctly.

5.27 Available DALIBs

ADAPTOR offers several DALIBs for each target machine, because it is available for different communication models on the target machines.

By default, the different DALIBs (rsp. UNILIBs) are located in the directory

```
$PHOME/dalib/<MP>/dalib.a           for SPMD programs
$PHOME/dalib/<MP>/unilib.a          for serial programs
```

<MP> stands for the communication model. Available communication models are listed in Table 1.

<MP>	related communication model
PVM3	PVM
MPI	MPI
SHM	communication via a System V shared memory segment
SP	native communication of IBM SP (POE, EUI interface)
PGON	native communication of Intel PARAGON (NX/2)
CS	native communication of Meiko CS (NX/2)
CM5	native communication of Thinking Machines CM5 (CMMD)
EPX	communication API of Embedded Parix

Table 1: Supported message passing systems for the DALIB.

6 The Compiler Driver gmdhpf

ADAPTOR provides its own compiler driver called `gmdhpf`.

`gmdhpf` is composed of a compile step and a link step, the compile step consisting of cpp-step, fadapt-step and fortran-compilation-step. Each of the steps is invoked depending on the kind of inputfiles, which is recognized by the suffix of the filenames as described below.

In *compile step* `gmdhpf` proceeds the inputfiles one after the other, that means first the first inputfile is processed through all the processors needed, depending upon the suffix of the inputfile, f.e.:

> first `inputfile1.HPF` is passed through cpp generating `inputfile1.hpf`;
> then `inputfile1.hpf` is passed through `fadapt` generating `inputfile1.f`;
> then `inputfile1.f` is passed through fc generating `inputfile1.o`

Hereafter the next inputfile is processed and so on until all the inputfiles have been processed; inputfiles for ld are just queued into a list of inputfiles for the linker.

If no error occurred at any time in compile step and if `gmdhpf` is not called with command-line option `-c` the *link step* is invoked.

Only the link step is invoked if there are just inputfiles with linker-suffixes.

`gmdhpf` exits (-1) if any error occurred and (0) else.

The names of the processors and their command line options are predefined. These predefinitions can be overridden by redefinitions in configuration files as described in section 7 below. The options given via gmdhpf command-line option `-W...` are added to the command-line options of the related processor valid after interpretation of the configuration files.

6.1 Dry Run

The option `dryrun` will show but not execute all the commands of the compile driver. It will also verify that the related libraries (HPF runtime system, message passing library, user libraries) are all available.

```
    gmdhpf -dryrun prime.hpf

submitting: /home/adaptor/bin/fadapt -f  prime.hpf
submitting: f77 -c -O -e -f -w prime.f -o prime.o
submitting: rm -f prime.f
submitting: f77 -o ./adaptor.out -g -lnsl -lsocket prime.o
            /home/brandes/adaptor/dalib/PVM3/dalib.a
            $PVM_ROOT/lib/$PVM_ARCH/libpvm3.a
```

6.2 Verbose and Help Information

Using the option `-help` gives a summary of all available options for the compiler driver.

```
    gmdhpf -help
```

The option `-v` for verbose shows all the commands executed by the compile driver.

```
    gmdhpf -v prime.hpf
```

```
/home/brandes/adaptor/bin/fadapt -f  prime.hpf
  f77 -c -O -e -f -w prime.f -o prime.o
  prime.f:
    MAIN prime:
  rm -f prime.f
  f77 -o a.out prime.o /home/brandes/adaptor/dalib/PVM3/dalib.a \
       $PVM_ROOT/lib/$PVM_ARCH/libpvm3.a -lnsl -lsocket
```

6.3 The Settings of the Compiler Driver

The following command gives information about the compilers and linkers that will be used by the compile driver `gmdhpf`.

```
gmdhpf -settings
```

The output of this command looks like this:

```
Default settings read in from following configuration files:
./.adprc

ADAPTOR Home Directory : /home/brandes/adaptor
target_machine:        : Sun Solaris 2.x
target_communication   : PVM 3.x
DALIB:                 : /home/brandes/adaptor/dalib/PVM3/dalib.a
Mess.-Pass.-rout. in   : $PVM_ROOT/lib/$PVM_ARCH/libpvm3.a

Verbose = 0,    Dryrun = 0,   Keep = 0

invoke cpp for cpp input-files
             CPP       : cpp
             CPP_OPT   : -P

invoke fadapt for fadapt input-files
             FADAPT    : /home/brandes/adaptor/bin/fadapt
             FADAPT_OPT:

invoke fc for fc input-files
             FC        : f77
             FC_OPT    : -O -e -f -w

LINK STEP WILL BE INVOKED if no error occurrs at compilation step.
LD will be taken for generation of SPMD program a.out

Generation of SPMD program with:
             LD        : f77
             LD_OPT    : -lnsl -lsocket

Generation of SINGLE NODE program with:
             LD1       : f77
             LD1_OPT   : -lnsl -lsocket
```

On some machines it might be necessary to have different linkers for the parallel SPMD and the serial program. Therefore two different linkers `LD` and `LD1` can be specified as well as specific options for these linkers.

Section 7 explains how these settings can be changed by configuration files.

6.4 Inputfiles for gmdhpf

cpp Files suffixed by

`.HPF, .F90, .F9, .H90 or .FCM`

are considered to be inputfiles for cpp.

For each of them cpp is invoked, generating an inputfile for `fadapt` if no error occurred, its name being the same as the name of the inputfile for cpp but with lowercase suffix. After this cpp step the generated fadapt inputfile is processed by `fadapt`.

Attention: Please prevent `cpp` from producing "line control information used in the next step of the C-Compiler"; it would disturb `fadapt`. E.g. for Solaris use cpp-option `-P` (is predefined).

fadapt Files suffixed by

`.hpf, .f90, .f9, .h90 or .fcm`

are considered to be inputfiles for `fadapt`.

For each of them `fadapt` is invoked, generating an inputfile for a native fortran compiler if no error occurred, its name being the same as the name of the inputfile for `fadapt` but with suffix `.f`. After this the generated Fortran file is processed by a native Fortran compiler.

Fortran compilation Files suffixed by

`.f, .F, .for, .FOR, .e, .r, .s, .S, .il or .tcov`

are considered to be inputfiles for a native Fortran compiler.

For each of them the native fortran compiler chosen by the user (see section 7) is invoked, generating an inputfile for ld or ld1 if no error occurred, its name being the same as the inputfile for fc but with suffix `.o`.

Linking Files suffixed by

`.o or .a`

are considered to be inputfiles for a native linker.

If the compile step has been completed without errors and if the link step is not cancelled by the user (option `-c` at `gmdhpf` call), all the generated object files and explicitly mentioned ld input files are linked together. Usually, the link step of the native Fortran compiler is invoked, but the user can also select other linkers (see section 7). The linker generates an executable named `./a.out` - if the user does not choose another name calling `gmdhpf` with option `-o <outfile name>`.

By default, `gmdhpf` will invoke the linker `LD` in order to generate an SPMD program. Just call `gmdhpf` with the option `-1` if you intend to generate a serial executable, and it will be done invoking linker LD1.

6.5 Intermediate Files

Intermediate files are generated by `gmdhpf` in the same directory as the inputfile. Unless the user specifies `-keep` when invoking gmdhpf, they are removed as soon as they are no longer needed. E.g., let `file1.HPF` be an inputfile and no error occured at compilation time:

- cpp output file `file1.hpf` is removed after fadapt step

- fadapt output file `file1.f` is deleted after fortran compilation step.
- The fortran output file `file.o` will *not* be erased.

6.6 Simple Options

With the option `-c` the link step is disabled. The input files will only be compiled but not linked.

```
gmdhpf -c prime.hpf
```

With the option `-o <outfile>` it is possible to chose the name for the final executable.

```
gmdhpf -o prime prime.o
```

6.7 Passing Options to Compiler and Linker

The `gmdhpf` flag `-W` enables the user to pass options to the different phases of the compilation:

`-Wcpp"options"` will pass options to the preprocessor:

```
gmdhpf -Wcpp"<cpp_options>" prime.HPF
```

`-Wa"options"` wil pass options to `fadapt`:

```
gmdhpf -Wa"<fadapt_options>" prime.hpf
```

`-Wf"options"` will to pass options to the Fortran compiler. This might be very useful to take full advantage of the best optimizations that the compiler provides.

```
gmdhpf -Wf"<fc_options>" prime.hpf
```

`-Wl"options"` will to pass options to the linker:

```
gmdhpf -Wl"<ld_options>" prime.hpf
gmdhpf -dryrun -Wl"-L/home/blas/lib -lblas3" <file>.hpf
```

7 Configuration File for gmdhpf

The configuration files for `gmdhpf` may be used to reset the predefined names of and options for the invoked prepocessor, `fadapt`, fortran-compiler and linker as well as the names of the `DALIB` and the message passing libraries used for linking.

Thus each user can easily change the preassumptions of the ADAPTOR installation according to his own needs.

A configuration file for `gmdhpf` must be named `".adprc"`. `gmdhpf` interprets such configuration files and looks for them in certain directories in the following order:

- `$PHOME/.adprc`

  This is the usual place to configure the compiler driver for the ADAPTOR installation on a certain machine.

- `$HOME/.adprc`

  This is the usual place where every user can make its own definitions (e.g. if he wants to use the Fortran 90 compiler instead of the FORTRAN 77 compiler by default).

- `./.adprc`

  If only a specific application should be compiled with certain flags, the configuration file can be put in the directory where the program files are located and compiled.

The settings of each configuration file replace previously valid settings.

If an option is not mentioned in any configuration file or if no configuration file was found at all, the default values from the ADAPTOR installation are taken.

The options for each processor given via `-W` at gmdhpf call are added to the ones valid after reading the `.adprc` file.

To get the settings valid for a `gmdhpf` run, type

```
gmdhpf -settings
```

7.1 Syntax of Entries in the Configuration File

Entries in .adprc files have the form:

```
<identifier> = <value>     or
#<this is a comment line>
```

Leading and trailing blanks are ignored for identifier and value.

Note: For the identifier on the left hand side upper or lower case letters are not significant. But they are significant for the value on the right hand side.

If `gmdhpf` detects an unknown identifier, the whole line is ignored.

7.2 Possible Definitions in Configuration File

Here you see all the possible definitions for the configuration file. Right-hand of the '=' the default values are shown, which may be overridden by redefinition in the installation configuration file `$PHOME/.adprc` or in your own configuration files `$HOME` and `./.adprc`.

```
# gmdhpf compiler driver configuration file

# specification of target machine (sets default compilers)
# e.g. CM5, PGON, CS, KSR, RS6K, SUN4, SUN4SOL2, SGI, HPPA, ALPHA
#      PPC, CENJU3, CRAYMPP

target_machine       = SUN4SOL2

# specification of communication model (sets default DALIB/UNILIB)
# e.g. PVM, MPI, SHM, NATIVE

target_communication = PVM

dalib                = $PHOME/dalib/PVM3/dalib.a
cpp                  = ...
cpp_opt              = ...
fc                   = ...
fc_opt               = ...
ld                   = ...
ld_opt               = ...
ld1                  = ...
ld1_opt              = ...

fadapt               = $PHOME/bin/fadapt
fadapt_opt           = -strip 132

mpi_lib              = $MPI_ROOT/mpich/lib/$ARCH/ch_p4/libmpi.a
pvm_lib              = $PVM_ROOT/lib/$PVM_ARCH/libpvm.a
shm_lib              = $SHM_ROOT/lib/$ARCH/libshm.a
```

8 Execution of HPF Programs

8.1 Start of the Processes

Starting the parallel programs is dependent upon your parallel platform; some examples are shown in section 8.4.

For the SPMD programming model the parallel executable must be loaded on all node processors.

A serial executable (compiled with the option `-1`) is started in the usual way.

8.2 Number of Node Processes

The default number of node processes is usually the number of processors that has been reserved for running the parallel application. If such a reservation does not exist, there will be no default value.

The number of nodes can also be specified by giving an explicit argument or by setting the environment variable NP.

```
a.out 12         ! will start 12 node processes

setenv NP 12
a.out            ! will start 12 node processes
```

If there is no default value and no explicit specification for the number of processes, this number will be interactively asked for until a legal value has been given as an input. This is also done if there was an illegal explicit specification, e.g. if the number of node processes is bigger than the number of available nodes.

Instead of a single number, it is also possible to define the number of processors as a product of two or three numbers. The corresponding processor shape will then be two- or three-dimensional.

```
a.out 3x4       ! starts 12 nodes as a 3x4 processor topology
a.out 2x2x2     ! starts 8 node processes as a 2x2x2 processor topology
```

Also the actual mapping of arrays distributed in two or three dimensions onto the default processor array can be driven at runtime by specifying the number of processors as a product.

```
      real A(N,N)
!hpf$ distribute A(block,block)

      a.out 1x4       ! only last dimension of A is distributed
      a.out 4x1       ! only first dimension of A is distributed
      a.out 2x2       ! both dimensions of A are distributed
```

8.3 Runtime Options

Some runtime options are provided to give some more information about the parallel HPF program during the runtime. This information can be very helpful for performance tuning of the HPF program. Table 5 in the appendix gives a summary of all available options. The flag `-help` prints also a summary of thes runtime options.

```
a.out -help

<executable> [-call] [-redist] [-comm] [-time] [-t] [n1[xn2[xn3]]]
```

- The flag `-call` allows to see which routines are entered and left.
- The flag `-redist` informs when a redistribution takes place. It will also give information when communication schedules for indirect addressing are reused.
- If the flag `-comm` has been set, a communication statistics will be printed at the end of the program.
- If the flag `-time` has been set, timing information for the different subprograms and nodes will be collected and printed into the file `<executable>.timestat`.
- The flag `-t` enables the generation of a tracefile to be used for visualization with VAMPIR (only available with MPI).

Instead of using comand line arguments it is also possible to set corresponding environment variables.

```
setenv CALL   1|on|ON
setenv REDIST 1|on|ON
setenv TIME   1|on|ON
setenv COMM   1|on|ON
setenv TRACE  1|on|ON
```

The communication statistics gives information about how many bytes of data have been sent and received between all processors. More detailed information is printed in the file <executable>.commstat.

```
STATISTICS SUMMARY:
Number of bytes sent by all procs:
 to:     P 1     P 2     P 3     P 4      all
 P 1    8000   31600   31200     400    71200
 P 2    8400       0     400       0     8800
 P 3    8000     400       0   31600    40000
 P 4    7600       0     400       0     8000
```

8.4 Starting Parallel Programs

8.4.1 PVM

PVM is a software system that permits a heterogeneous collection of Unix computers networked together to be viewed by a user's program as a single parallel computer [GBD+94]. As the PVM software has been distributed freely, it is being used all around the world.

Before starting the parallel executable, the PVM daemon must be running on all machines that will be used for the parallel execution. Afterwards the executable can be started directly from a Unix prompt on any host of the virtual machine.

```
a.out 5
```

It should be noted that at the beginning only one process is started. During the initialization phase, this process spawns itself until the specified number of processes is achieved (specified as a runtime argument or with the environment variable NP). The parallel executable must have been copied to the appropriate directories (usually $PVM_ROOT/bin/$ARCH) before.

Note: ADAPTOR does not support heterogeneous architectures at this time. Therefore, only machines of one architecture should be used for the parallel execution.

8.4.2 MPI

The Message-Passing Interface (MPI) standard has been introduced as a common standard for the message passing programming paradigm to achieve portability between different parallel machines. Since almost all major hardware vendors were actively involved in MPI, implementations are meanwhile available on virtually every parallel computer, including networks of workstations.

MPI programs usually are started by the mpirun command:

```
mpirun -np <np> -arch <arch> a.out
```

8.4.3 Intel iPSC/860

Before starting a parallel program on the iPSC, the user has to attach to a cube with the command **getcube**.

```
getcube -tn   (reservation of a cube with n processors)
```

The user has to load explicitly the node program on all nodes.

```
load a.out;  waitcube
```

It is possible to limit the number of node processes to a number smaller than the number of reserved nodes by using the environment variable `NP`.

After the termination of the parallel program the cube should be released by using the command **relcube**.

```
relcube
```

8.4.4 Intel Paragon XP/S

The parallel program on the Paragon is started in the same way as every usual parallel program on this machine.

```
pexec a.out -sz 16
```

Another solution for running the parallel program is the following one:

```
novpart -sz 16 mypart
a.out -pn mypart
```

By using the environment variable `NP` it is possible to run the program on less nodes than the number of nodes in the reserved partition.

8.4.5 Meiko CS

The realization of the DALIB for the Meiko CS machines is based on their emulation of Intels NX libraray. Please have a look at the manuals to find out the specific commands for running programs on this machine.

8.4.6 CM-5

A parallel program will be started on a partition of the CM-5. The default number of processors is the number of processors of the reserved partition. The number of actual used processors can be smaller than the number of these processors, but it is not possible to have more processes.

Currently CM Fortran is not supported as target language. Therefore it is not possible to generate CM Fortran programs with message passing. The only way to use the CM-5 is to generate Fortran 77 programs and to use a SUN Fortran 77 compiler with the CMMD message passing library.

8.4.7 IBM SP

Compiling, linking and running the parallel programs is done by using the features of the IBM AIX Parallel Environment (PE).

```
setenv MP_PROCS 4
a.out
```

8.4.8 Cray T3D and Cray T3E

```
mppexec -npes 4 a.out
a.out -npes 4

mpprun -n 4 a.out
```

9 Parallel Execution of Different HPF Tasks

ADAPTOR allows the concurrent execution of multiple HPF tasks that might be coupled via the new HPF_TASK_LIBRARY (see figure 2). Every HPF task can be an own HPF program.

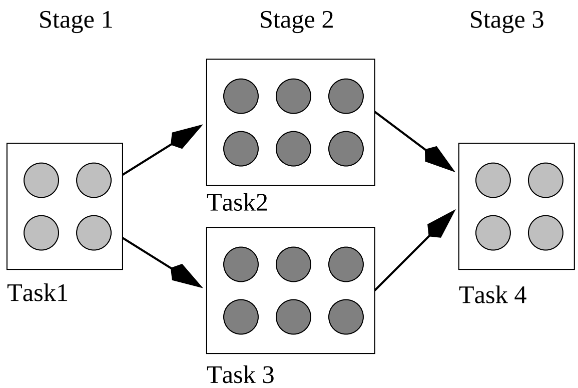


Figure 2: Concurrent execution of data parallel tasks.

The coupling of multiple HPF programs is only possible when all node processes are started as one application (MPMD execution model). The node processes belonging to one HPF task will build a new subgroup that becomes the actual context of this task program. Every task and every processor will know all subgroups and so all available HPF tasks. This is needed for corresponding communication between the HPF tasks. Such routines are provided by the HPF_TASK_LIBRARY.

9.1 Generation of HPF Tasks

HPF programs that should become HPF tasks are compiled in the usual way. There is no special flag for compilation needed as the multiple execution of HPF tasks is completely hidden in the DALIB runtime system.

```
gmdhpf -o STAGE1 STAGE1.HPF
gmdhpf -o STAGE2 STAGE2.HPF
gmdhpf -o STAGE3 STAGE3.HPF
```

The usual way to guarantee the parallel execution of HPF tasks is to create a task file (e.g. TASKS) that contains the corresponding task names and task sizes. Comment lines are allowed for this file. The environment variable TASK_FILE must be set with the name of the task file. This environment variable will be checked at runtime and the corresponding creation of HPF tasks initiated.

```
setenv TASK_FILE TASKS

# content of task file TASKS
STAGE1  2
STAGE2  3
STAGE2  3
STAGE3  2
```

For the given example, four HPF tasks should be executed simultaneously. There will be two different HPF tasks for STAGE2 with three processors for every task. The other two tasks, STAGE1 and STAGE3, will run on two processors in each case.

The parallel machine itself must provide a mechanism to start the different executables on the nodes of the parallel machine. The numbering and execution of the executables must correspond to then entries in the task file.

If the environment variable TASK_FILE has not been set, the DALIB will also build subgroups of different HPF tasks as soon as the loaded executables have different names. The algorithm for building subgroups is based on the following idea: all processors with the same name of the loaded program will form a new subgroup. Every such subgroup forms one HPF task. This allows the multiple execution of different HPF programs. If the same HPF program should run for different tasks, the executable must be renamed.

9.2 PVM

The DALIB implementation based on PVM spawns the processes on its own. Therefore it is sufficient to define a corresponding task file (e.g. TASKS) that contains the corresponding task names and task sizes to set environment variable TASK_FILE with the filename. When starting the executable of the first task, the other processes will be started automatically.

```
setenv TASK_FILE TASKS
STAGE1
```

It is mandatory that the called executable is part of the first task. The user has also to guarantee that the PVM daemons find the corresponding executables of all HPF tasks (e.g. STAGE1, STAGE2, STAGE3). This is described in the PVM User Guide.

9.3 MPI

Some MPI implementations allow the execution of different SPMD programs in the MPMD model. E.g., MPICH (portable implementation of MPI) is based on P4 and allows the definition of a processor group file (e.g. PGFILE).

```
Machine01 0 STAGE1
Machine02 1 STAGE1
Machine03 1 STAGE2
Machine04 1 STAGE2
Machine05 1 STAGE2
Machine06 1 STAGE2
Machine07 1 STAGE2
Machine08 1 STAGE2
Machine08 1 STAGE3
Machine09 1 STAGE3
```

The task file and the processor group file must define the tasks in the same order.

```
setenv TASK_FILE TASKS
mpirun -p4pg PGFILE STAGE1
```

Note: When starting the same processes without setting the environment variable TASK_FILE, the different executables will also be executed as HPF tasks. But in this case, all processes running STAGE2 will form one single HPF task running on 6 processors.

9.4 IBM SP2

For loading and executing different HPF programs the command poe should be used that invokes the Parallel Operation Environment (POE). As different executables should be loaded, a corresponding command file (e.g. CMDFILE) containing the names of the executables to be started must be defined.

```
STAGE1
STAGE1
STAGE2
STAGE2
STAGE2
STAGE2
STAGE2
STAGE2
STAGE3
STAGE3
```

Afterwards, the poe command is called in the following way:

```
setenv TASK_FILE TASKS
poe -cmdfile CMDFILE -pgmmodel MPMD
```

The same can be achieved with the following commands:

```
setenv TASK_FILE TASKS
setenv MP_PGMMODEL MPMD
setenv MP_CMDFILE CMPDFILE
poe
```

10 Problems

If any problems occur while using ADAPTOR, please mail these problems to the authors. We will try to solve them as soon as possible.

For our future work, it is also important to get as much feedback as possible. Only due to the feedback we got during the last years, this current release could be made available. Unfortunately, we were not able to take all suggestions into account. But we will try to consider them for future releases.

References

[BHK98] T. Brandes and R. Höver-Klier. ADAPTOR Installation Guide (Version 6.0). Technical documentation, GMD, June 1998. Available via anonymous ftp from `ftp.gmd.de` as `gmd/adaptor/docs/iguide.ps`.

[Bra97] T. Brandes. Compiling High Performance Fortran for Shared Memory and Shared Virtual Memory Systems. In University of Westminster, editor, *Proceedings of the 5th Euromicro Workshop on Parallel and Distributed Computing, London, UK*, pages 289–296, London, UK, January 1997. IEEE Computer Society Press.

[Bra98] T. Brandes. ADAPTOR Programmer's Guide (Version 6.0). Technical documentation, GMD, June 1998. Available via anonymous ftp from `ftp.gmd.de` as `gmd/adaptor/docs/pguide.ps`.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* Scientific and Engineering Computation Series. The MIT Press, Cambridge, MA, 1994.

[Hig97] High Performance Fortran Forum. High Performance Fortran Language Specification. Version 2.0, Department of Computer Science, Rice University, January 1997.

[SGI97] SGI. OpenMP Fortran Application Program Interface. Proposal Ver 1.0, SGI, October 1997.

Appendix

Table 2 and 3 show a summary of all available options of the source-to-source translator `fadapt`.

-HPF_SUBSET	source language = HPF (subset)
-HPF_BASE	source language = HPF 2.0 Base Language
-HPF	source language = HPF 2.0 (full)
-HPF_ADP	source language = HPF 2.0 + ADAPTOR extensions
-F77	target language = FORTRAN 77
-F90	target language = Fortran 90
-auto	enables automatic loop parallelization
-noauto	disables automatic loop parallelization
-N	model = SPMD prog
-1	model = Uni Proc
-ddr	default distribution = replicated
-ddb	default distribution = block distr
-d <n>	maximal number of distributed dimensions
-strip <n>	number of columns in generated Fortran sources
-fix <n>	number of relevant columns in input file
-free	free source format
-f <filename>	name of output file (cube.f or node1.f is default)
-v	verbose
-debug	generate intermediate files xxx_unparse.f
-list	keep protocol files adaptor.xxx
-G	abbreviation for -v -debug -list
-version	print current version of fadapt
-w	suppresses warning messages
-noinfo	suppresses info messages
-I<include_dir>	specify additional include directory
-interface	Generate Interface Specification
-call	Generate Call Graph
-i4 or -i8	default size of integer is 4 or 8 bytes
-r4 or -r8	default size of real is 4 or 8 bytes
-a4 or -a8	default size of address is 4 or 8 bytes
-sp or -dp	REAL is single / double precision
-O	optimization
-noopt	no optimizations
-safety <n>	run time checking (n=0, 1 or 2)
-nostrict	no check of arguments

Table 2: Available options for `fadapt`.

-novector	generate code optimized for cache
-vector	generate code optimized for vectorization
-mp=no	generates code without parallelization directives
-mp	generates code with Open MP parallelization directives
-mp=sun	generates code with Sun MP parallelization directives
-mp=sgi	generates code with SGI MP parallelization directives
-mp=cray	generates code with Cray MP parallelization directives
-shared	distributed arrays without any replication are shared
-static	static distributed arrays will not become dynamic

Table 3: Machine specific options for `fadapt`.

Table 4 describes the options for the compile driver `gmdhpf`.

-help	prints help for available options
-settings	prints the actual settings
-v	verbose mode
-dryrun	shows commands without executing them
-c	suppress link step
-1	serial compilation of HPF programs
-o <outfile>	renaming of executable (instead of `a.out`)
-keep	do not remove intermediate files
-Wcpp"[cpp_opt]"	passing options to cpp preprocessor
-Wa"[fadapt_opt]"	passing options to fadapt compilation
-Wf"[fortran_opt]"	passing options to Fortran compilation
-Wl"[link_opt]"	passing options to the linker

Table 4: Available options for `gmdhpf`.

Table 5 summarizes the available runtime options of parallel ADAPTOR executables.

-help	print help for available runtime options
-call	info when entering and leaving routines
-redist	info about possible redistributions
-comm	prints communication statistics
-time	prints timing statistics
-t	generates trace file for VAMPIR
<n>	execution on n processors
<n1>x<n2>	execution on two-dimensional processor topology
<n1>x<n2>x<n3>	execution on three-dimensional processor topology

Table 5: Available runtime options for executables.

Index